

# A DNF Blocking Scheme Learner for Heterogeneous Datasets

## ABSTRACT

Entity Resolution concerns identifying co-referent entity pairs across datasets. A typical workflow comprises two steps. In the first step, a blocking method uses a one-many function called a blocking scheme to map entities to blocks. In the second step, entities sharing a block are paired and compared. Current DNF blocking scheme learners (DNF-BSLs) apply only to structurally homogeneous tables. We present an unsupervised algorithmic pipeline for learning DNF blocking schemes on RDF graph datasets, as well as structurally heterogeneous tables. Previous DNF-BSLs are admitted as special cases. We evaluate the pipeline on six real-world dataset pairs. Unsupervised results are shown to be competitive with supervised and semi-supervised baselines. To the best of our knowledge, this is the first unsupervised DNF-BSL that admits RDF graphs and structurally heterogeneous tables as inputs.

## Categories and Subject Descriptors

I.2.6 [Artificial Intelligence]: Learning; H.2.8 [Database Management]: Database applications—*Data Mining*

## General Terms

Algorithms, Experimentation, Performance

## Keywords

Heterogeneity, Entity Resolution, Unsupervised Learning

## 1. INTRODUCTION

*Entity Resolution* (ER) is the identification of co-referent entities across datasets. Different communities refer to it as *instance matching*, *record linkage* and the *merge-purge problem* [12, 13]. Scalability indicates a two-step solution [12]. The first step, *blocking*, mitigates brute-force pairwise comparisons on all entities by clustering entities into blocks and then comparing pairs of entities only within blocks [9].

Blocking results in the selection of a small subset of pairs, called the *candidate* set, which is input to a second step to determine co-reference using a sophisticated similarity function. We exclusively address blocking in this work.

Blocking methods use a *blocking scheme* to assign entities to blocks. Recently, Disjunctive Normal Form (DNF) Blocking Scheme Learners (BSLs) were proposed to learn DNF blocking schemes using supervised [3, 21], semi-supervised [6] or unsupervised machine learning techniques [16].

DNF-BSLs operate in an expressive hypothesis space and have shown excellent empirical performance [3]. Despite their advantages, current DNF-BSLs assume that input datasets are *tabular* and have the *same schemas*. The latter assumption is often denoted as *structural homogeneity* [12].

The assumption of tabular structural homogeneity restricts application of DNF-BSLs to other data models. Recent growth of graph datasets and Linked Open Data (LOD) [5] motivates the development of a DNF-BSL for the RDF<sup>1</sup> data model. These graphs are often published by independent sources and are *heterogeneous* [27].

In this paper, we present a *generic algorithmic pipeline* for learning DNF blocking schemes on pairs of RDF datasets. The formalism of DNF blocking schemes relies on the existence of a *schema*. RDF datasets on LOD may not have accompanying schemas [5]. Instead of relying on possibly unavailable metadata, we build a *dynamic schema* using the properties in the RDF dataset. The RDF dataset can then be *logically* represented as a *property table*, which may be populated at run-time. Previously, property tables were defined as *physical* data structures used in the implementation of *triplestores* [28]. Using a logical property table representation admits application of a DNF-BSL to RDF datasets.

As a special case, the pipeline also admits *structurally heterogeneous* tabular inputs. That is, the pipeline can be applied to tabular datasets with *different* schemas. Thus, previous DNF-BSLs become special cases of the pipeline, since they learn schemes on tables with the same schemas. As a second special case, the pipeline accommodates *RDF-tabular heterogeneity*, with one input, RDF, and the other, tabular. RDF-tabular heterogeneity applies when linking datasets between LOD and the relational Deep Web [15, 5].

Much existing ER research bypasses the issue of structural heterogeneity by assuming that *schema matching* is physically undertaken prior to executing an ER algorithm [12]. Schema matching itself is an active, challenging research problem [1, 14]. Instead of relying on the unrealistic assumption of perfect schema reconciliation, the pipeline

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

<sup>1</sup>Resource Description Framework

admits possibly noisy schema mappings from an existing schema matcher, and *directly* uses the mappings to learn a DNF blocking scheme. If a matcher is unavailable, the pipeline allows for a *fall-back* option.

We show an *unsupervised instantiation* of the generic pipeline by strategically employing an existing instance-based schema matcher called *Dumas* [4]. In addition to schema mappings, Dumas also generates noisy duplicates, which we *recycle* and pipe to a *new* DNF-BSL. The new DNF-BSL uses only two parameters, which may be robustly tuned for good empirical performance across multiple domains. The BSL is also shown to have a strong *theoretical* guarantee.

We evaluate the full unsupervised pipeline on six real-world dataset pairs against two extended baselines, one of which is *supervised* [3, 16]. Our system performance is found to be competitive, despite training samples not being manually provided. Given that unsupervised methods exist for the *second* ER step in both the Semantic Web and relational communities [13, 12], this first unsupervised heterogeneous DNF-BSL *enables*, in principle, fully unsupervised ER in *both* communities.

Section 2 describes some related work in the area, followed by preliminaries in Section 3. Section 4 describes the generic pipeline, followed by an unsupervised instantiation in Section 5. Section 6 describes the experiments, with results discussed in Section 7. The paper concludes in Section 8.

## 2. RELATED WORK

ER was comprehensively surveyed by Elmagarmid et al. [12], with a generic approach represented by *Swoosh* [2]. Separately, blocking has witnessed much specific research, with Christen surveying blocking methods [9]. Bilenko et al. [3], and Michelson and Knoblock [21] independently proposed supervised DNF-BSLs in 2006. Since then, a semi-supervised adaptation of the BSL proposed by Bilenko et al. has been published [3, 6], as well as an unsupervised system [16]. The four systems assume *structural homogeneity*. We discuss their core principles in Section 4.3.

Heterogeneous blocking may be performed without learning a DNF scheme. One example is Locality Sensitive Hashing (LSH), employed by the *Harra* system, for instance [17]. LSH is promising but applies only to specific distance measures, like Jaccard and cosine. The *Typifier* system by Ma et al. is another recent example that relies on *type inferencing* and was designed for Web data published without semantic types [20]. In contrast, DNF-BSLs can be applied *generally*, with multiple studies showing strong empirical performance [21, 3, 6, 16]. Finally, although related, *clustering* is technically treated separately from blocking in the literature [9].

In the Semantic Web, ER is simultaneously known as *instance matching* and *link discovery*, and has been surveyed by Ferraram et al. [13]. Existing works restrict inputs to RDF. Also, most techniques in the Semantic Web do not *learn* schemes, but instead present graph-based blocking *methods*, a good example being Silk [27].

Finally, the framework in this paper also relies on *schema mapping*. Schema mapping is an active research area, with a good survey provided by Bellahsene et al. [1]. Gal notes that it is a difficult problem [14].

Schema matchers may return 1:1 or n:m mappings (or even 1:n and n:1). An *instance-based* schema matcher relies on data instances to perform schema matching [1]. A good example is *Dumas* [4], which relies on an inexpensive *duplica-*

*tes generator* to perform unsupervised schema matching [4]. We describe Dumas in Section 5.

The *property table* representation used in this paper is a *physically* implemented data structure in the Jena triplestore API<sup>2</sup> [28]. In this paper, it is used as a *logical* data structure. We note that the concept of logically representing one data model as another has precedent. In particular, the literature abounds with proposed methods on how to integrate relational databases (RDB) with the Semantic Web. Sahoo et al. extensively surveyed this topic, called *RDB2RDF* [24]. A use-case is the *Ultrawrap* architecture, which utilizes RDB2RDF to enable real-time Ontology-based Data Access or OBDA [25]. We effectively tackle the *inverse* problem by translating an RDF graph to a logical property table. This is the first application to devise such an inverse translation for heterogeneous ER.

## 3. PRELIMINARIES

We present definitions and examples to place the remainder of the work in context. Consider a pair of datasets  $R_1$  and  $R_2$ . Each dataset individually conforms to either the RDF or tabular data model. An *RDF* dataset may be visualized as a *directed graph* or equivalently, as a set of *triples*. A triple is a 3-tuple of the form *(subject, property<sup>3</sup>, object)*. A *tabular* dataset conforms to a *tabular schema*, which is the table name followed by a set of *fields*. The dataset *instance* is a set of *tuples*, with each tuple comprising *field values*.

*Example 1.* Dataset 1 (in Figure 1) is an RDF dataset visualized as a directed graph  $G = (V, E)$ , and can be equivalently represented as a set of  $|E|$  triples. For example, *(Mickey Beats, hasWife, Joan Beats)* would be one such triple in the triples representation. Datasets 2 and 3 are tabular dataset examples, with the former having schema *Emergency Contact(Name, Contact, Relation)*. The first tuple of Dataset 2 has field values *Mickey Beats, Joan Beats* and *Spouse* respectively. The keyword *null* is reserved.

According to the RDF specification<sup>4</sup>, subjects and properties must necessarily be *Uniform Resource Identifiers* (URIs), while an object node may either be a URI or a literal. URI elements in RDF files typically have associated *names* (or *labels*), obtained through dereference. For ease of exposition, we henceforth refer to every URI element in an RDF file by its associated name. Note also, that in the most general case, RDF datasets do not have to conform to any schema. This is why they are commonly visualized as *semi-structured* datasets, and not as tables. In Section 4.1, we show how to dynamically build a *property schema* and logically represent RDF as a tabular data structure.

An entity is defined as a *semantically* distinct *subject* node in an RDF dataset, or as a (semantically distinct) *tuple* in a tabular dataset. The entity referring to *Mickey Beats* is shown in red in all datasets in Figure 1. *Entity Resolution* is the process of resolving *semantically equivalent* (but possibly *syntactically different*) entities. As earlier described, ER is traditionally conducted on tables with the same schemas, or on *one*<sup>5</sup> dataset. An example would be identifying that the two highlighted tuples in Dataset 3 are duplicates.

<sup>2</sup><https://jena.apache.org/>

<sup>3</sup>Alternatively called *predicate*; we uniformly use *property*.

<sup>4</sup><http://www.w3.org/RDF/>

<sup>5</sup>Commonly just called *deduplication*, if this is the case [12].

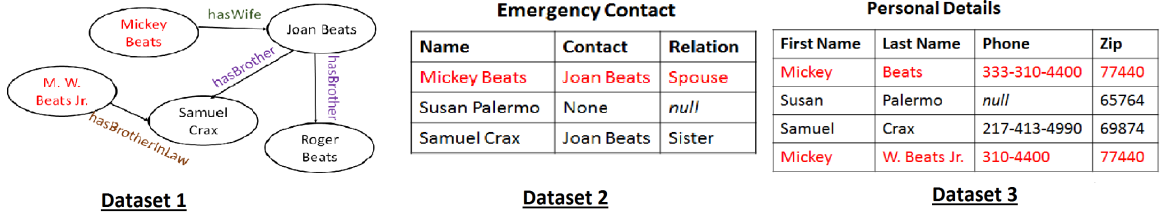


Figure 1: Three example datasets exhibiting various kinds of heterogeneity

In the Semantic Web, ER is operationalized by connecting two equivalent entities with an *owl:sameAs*<sup>6</sup> property edge. For example, the two nodes referring to Mickey Beats in Dataset 1 should be connected using an *owl:sameAs* edge. Easy operationalizing of ER (and more generally, *link specification* [27]) explains in part the recent interest in ER in the Semantic Web [13]. In the relational setting, ER is traditionally operationalized through joins or mediated schemas. It is less evident how to operationalize ER across RDF-tabular inputs, such as linking Datasets 1 and 2. We return to this issue in Section 4.1.

To introduce the *current* notion of DNF blocking schemes, tabular structural homogeneity is assumed for the remainder of this section. In later sections, we generalize the concepts.

The most basic elements of a blocking scheme are *indexing functions*  $h_i(x_t)$  [3]. An indexing function accepts a field value from a tuple as input and returns a set  $Y$  that contains 0 or more *blocking key values* (BKVs). A BKV identifies a block in which the tuple is placed. Intuitively, one may think of a block as a hash bucket, except that blocking is one-many while hashing is typically many-one [9]. For example, if  $Y$  contains multiple BKVs, a tuple is placed in multiple blocks.

**Definition 1.** An *indexing function*  $h_i : \text{Dom}(h_i) \rightarrow U^*$  takes as input a field value  $x_t$  from some tuple  $t$  and returns a set  $Y$  that contains 0 or more *Blocking Key Values* (BKVs) from the set of all possible BKVs  $U^*$ .

The domain  $\text{Dom}(h_i)$  is usually just the *string* datatype. The range is a set of BKVs that the tuple is assigned to. Each BKV is represented by a string identifier.

**Example 2.** An example of an indexing function is *Tokens*. When applied to the *Last Name* field value of the fourth tuple in Dataset 3, the output set  $Y$  is  $\{W., \text{Beats}, \text{Jr.}\}$ .

This leads to the notion of a *general blocking predicate* (GBP). Intuitively, a GBP  $p_i(x_{t_1}, x_{t_2})$  takes as input field values from two tuples,  $t_1$  and  $t_2$ , and uses the  $i^{\text{th}}$  indexing function to obtain BKV sets  $Y_1$  and  $Y_2$  for the two arguments. The predicate is satisfied *iff*  $Y_1$  and  $Y_2$  share elements, or equivalently, if  $t_1$  and  $t_2$  have a block in common.

**Definition 2.** A *general blocking predicate*  $p_i : \text{Dom}(h_i) \times \text{Dom}(h_i) \rightarrow \{\text{True}, \text{False}\}$  takes as input field values  $x_{t_1}$  and  $x_{t_2}$  from two tuples,  $t_1$  and  $t_2$ , and returns *True* if  $h_i(x_{t_1}) \cap h_i(x_{t_2}) \neq \Phi$ , and returns *False* otherwise.

Each GBP is always associated with an indexing function.

**Example 3.** Consider the GBP *ContainsCommonToken*, associated with the previously introduced *Tokens*. Suppose

<sup>6</sup><http://www.w3.org/TR/owl-ref/>

it was input the *Last Name* field values from the first and fourth tuples in Dataset 3. Since these field values have a token (*Beats*) in common, the GBP returns *True*.

A *specific blocking predicate* (SBP) explicitly pairs a GBP to a specific field.

**Definition 3.** A *specific blocking predicate* is a pair  $(p_i, f)$  where  $p_i$  is a general blocking predicate and  $f$  is a field. A specific blocking predicate takes two tuples  $t_1$  and  $t_2$  as arguments and applies  $p_i$  to the appropriate field values  $f_1$  and  $f_2$  from both tuples. A tuple pair is said to be *covered* if the specific blocking predicate returns *True* for that pair.

Previous DNF research assumed that all available GBPs can be applied to *all* fields of the relation [16, 3, 6, 21]. Hence, given a relation  $R$  with  $m$  fields in its schema<sup>7</sup>, and  $s$  GBPs, the number of SBPs is exactly  $ms$ . Finally, a *DNF blocking scheme* is defined as:

**Definition 4.** A *DNF blocking scheme*  $f_P$  is a *positive* propositional formula constructed in *Disjunctive Normal Form*<sup>8</sup> (DNF), using a given set  $H$  of SBPs as the set of *atoms*. Additionally, if each *term* is constrained to comprise at most one atom, the blocking scheme is referred to as *disjunctive*.

SBPs cannot be negated, since the DNF scheme is a positive formula. A tuple pair is said to be *covered* if the blocking scheme returns *True* for that pair. Intuitively, this means that the two constituent tuples *share* a block. In practice, both duplicate and non-duplicate tuple pairs can end up getting covered, since blocking is just a pre-processing step.

**Example 4.** Consider the disjunctive scheme  $(\text{ContainsCommonToken}, \text{Last Name}) \vee (\text{SameFirstDigit}, \text{Zip})$ , applied on Dataset 3. While the two tuples referring to Mickey Beats would share a block (with the BKV *Beats*), the non-duplicate tuples referring to Susan and Samuel would *also* share a block (with the BKV *6*). Note also that the first and fourth tuples share *more* than one block, since they also have BKV *7* in common.

Given a blocking scheme, a *blocking method* would need to map tuples to blocks *efficiently*. Per Definition 4, a blocking scheme takes a tuple pair as input. In practice, linear-time hash-based techniques are usually applied.

**Example 5.** To efficiently apply the blocking scheme in the previous example on *each individual* tuple, tokens from

<sup>7</sup>Structural homogeneity implies exactly one input schema, even if there are multiple relational instances.

<sup>8</sup>A disjunction of *terms*, where each term is a conjunction of literals.

the field value corresponding to field *Last Name* are extracted, along with the first character from the field value of the *Zip* field, to obtain the tuple's set of BKVs. For example, applied to the first tuple of Dataset 3, the BKV set  $\{Beats, 7\}$  is extracted. An index is maintained, with the BKVs as keys and tuple pointers as values. With  $n$  tuples, traditional blocking computes the blocks in time  $O(n)$  [9].

Let the set of generated blocks be  $\Pi$ .  $\Pi$  contains sets of the form  $B_v$ , where  $B_v$  is the *block* referred to by the BKV  $v$ . The *candidate set* of pairs  $\Gamma$  is given below:

$$\Gamma = \bigcup_{B_v \in \Pi} \{(r, s)\}, \forall r, s \in B_v | r \in R_1, s \in R_2 \quad (1)$$

$\Gamma$  is precisely the set input to the *second* step of ER, which classifies each pair as a duplicate, non-duplicate or probable duplicate [8]. Blocking should produce a small  $\Gamma$  but with high coverage and density of duplicates. Metrics quantifying these properties are defined in Section 6.

Finally, *schema mapping* is utilized in the paper. The *formal* definition of a mapping is quite technical; the survey by Bellahsene et al. provides a full treatment [1]. In this paper, an intuitive understanding of the mapping as a *pair* of *field-sets* suffices. For example,  $(\{Name\}, \{First Name, Last Name\})$  is a 1:n mapping between Datasets 2 and 3. More generally, mappings may be of cardinality n:m. The simplest case is a 1:1 mapping, with singleton components.

## 4. THE GENERIC PIPELINE

Figure 2 (a) shows a schematic of the *generic* pipeline. Two heterogeneous datasets are initially provided as input, with either dataset being RDF or tabular. If the dataset is RDF, we logically represent it as a *property table*. We describe the details and advantages of this *tabular* data structure in Section 4.1. The key point to note is that the *schema matching* module takes two *tables* as input, regardless of the data model, and outputs a set of schema mappings  $Q$ . An *extended* DNF-BSL accepts  $Q$  and also a training set of duplicates and non-duplicates as input, and learns an extended DNF blocking scheme. The DNF-BSL and the blocking scheme need to be extended because tables are now structurally heterogeneous, and the Section 3 formalism does not natively apply. In Section 4.2, the formalism is extended to admit structural heterogeneity.

Figure 2 (b) shows an *unsupervised instantiation* of the generic pipeline. We describe the details in Section 5.

### 4.1 Property Table Representation

Despite their formal description as sets of triples, RDF files are often *physically* stored in triplestores as sparse property tables [28]. In this paper, we adapt this table instead as a *logical* tabular representation of RDF. To enable the logical construction on RDF dataset  $R$ , define the *property schema*  $\{subject\} \cup \{p | \exists (s, p, o), (s, p, o) \in R\}$ . In essence, we flatten the graph by assigning each distinct *property* (or edge label) a corresponding field in this schema, along with an extra *subject* field. Every distinct subject in the triples-set has exactly one corresponding tuple in the table.

For example, Figure 3 is the property table representation of Dataset 1 in Figure 1. If a subject does not have a corresponding object value for a given property, the reserved keyword *null* is entered. If a subject has *multiple* object values for a property, the values are concatenated using a

Subject	hasWife	hasBrother	hasBrotherInLaw	← Property Schema
Mickey Beats	Joan Beats	null	null	
Joan Beats	null	Roger Beats; Samuel Crax	null	
M. W. Beats Jr.	null	null	Samuel Crax	

Figure 3: Property Table representation of Dataset 1 in Figure 1

reserved delimiter (; in Figure 3). Technically, field values now have *set* semantics, with *null* representing the empty set. Also, the original set of triples can be losslessly reconstructed from the property table (and vice versa), making it an *information-preserving* logical representation. Although intuitively straightforward, we detail these lossless conversions in the Appendix.

The *physical* property table was proposed to eliminate expensive *property-property self-joins* that occur frequently in SPARQL<sup>9</sup> queries. For ER, the logical data structure is useful because it allows for a *dynamic schema* that is resolvable at *run-time*. Triplestores like Jena allow updating and querying of RDF *graphs*, despite the underneath tabular representation [28]. If the RDF dataset is already stored in such a triplestore, it would not have to be moved prior to ER. This gives the property table a *systems-level* advantage.

More importantly, having a schema for an RDF dataset means that we can invoke a suitable schema matcher in the generalized pipeline. As we subsequently show, the extended DNF-BSL in the pipeline requires the datasets to have (possibly different) schemas<sup>10</sup>. Finally, representing RDF as a table allows us to address *RDF-tabular heterogeneity*, by reducing it to tabular structural heterogeneity. Traditional ER operations become well-defined for RDF-tabular inputs.

One key advantage of the property schema is that it does not rely on RDF Schema (RDFS) metadata. In practice, this allows us to represent *any* file on Linked Open Data tabularly, regardless of whether metadata is available.

Even in the simple example of Figure 3, we note that the property table is not without its challenges. It is usually *sparse*, and it may end up being *broad*, for RDF datasets with many properties. Furthermore, properties are *named* using different conventions (for example, the prefix *Has* occurs in all the properties in Figure 3) and could be *opaque*, depending on the dataset publisher. We show empirically (Section 6) that the *instantiated* pipeline (Section 5) can handle these difficulties.

### 4.2 Extending the formalism

The formalism in Section 3 assumed structural homogeneity. We extend it to accommodate structural heterogeneity. As input, consider two datasets  $R_1$  and  $R_2$ . If either dataset is in RDF, we assume that it is in property table form. Consider the original definition of SBPs in Definition 3. SBPs are associated with a *single* GBP and a *single* field, making them amenable only to a single schema. To account for heterogeneity, we extend SBPs by replacing the *field* input with a *mapping*. Denote as  $A_1$  and  $A_2$  the respective sets of fields of datasets  $R_1$  and  $R_2$ . We define *simple extended* SBPs below:

<sup>9</sup><http://www.w3.org/TR/rdf-sparql-query/>

<sup>10</sup>Even non-DNF blocking typically assumes this [20].

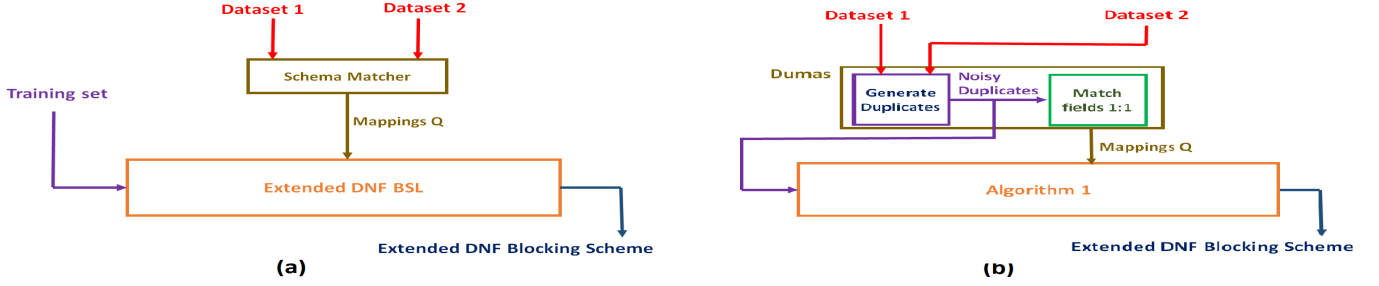


Figure 2: (a) shows the generic pipeline for learning a DNF blocking scheme on heterogeneous datasets, with (b) showing an *unsupervised instantiation*

*Definition 5.* A *simple extended specific blocking predicate* is a pair  $(p_i, m)$  where  $p_i$  is a general blocking predicate and  $m = (\{f_1\}, \{f_2\})$  is a mapping from a single field  $f_1 \in A_1$  to a single field  $f_2 \in A_2$ . The predicate takes two tuples  $t_1$  and  $t_2$  as arguments and applies  $p_i$  to the field values corresponding to  $f_1$  and  $f_2$  in the two tuples respectively.

The correspondence in Definition 5 is denoted as *simple* since it uses a mapping of the simplest cardinality (1:1). Definition 3 can be *reformulated* as a special case of Definition 5, with  $A_1 = A_2$  and  $f_1 = f_2$ .

The SBP semantics are not evident if the mapping cardinality is  $n:m$ ,  $1:n$  or  $n:1$ , that is, between two *arbitrary* field-subsets,  $F_1 \subseteq A_1$  and  $F_2 \subseteq A_2$ . If we *interpret* the two sets as representing  $|F_1||F_2|$  1:1 mappings, we obtain a *set* of simple extended SBPs  $\{(p_i, (\{f_1\}, \{f_2\})) | f_1 \in F_1, f_2 \in F_2\}$ .

The interpretation above is motivated by the requirement that an SBP should *always* return a boolean value. We approach the problem by using the  $n:m$  mappings to construct the set of simple extended SBPs, as shown above. We then use *disjunction* as a *combine* operator on all elements of the constructed set to yield a single boolean result. We can then define *complex extended* SBPs.

*Definition 6.* A *complex extended specific blocking predicate* is a pair  $(p_i, M)$  where  $p_i$  is a general blocking predicate and  $M$  is a mapping from a set  $F_1 \subseteq A_1$  to a set  $F_2 \subseteq A_2$ . The predicate takes two tuples  $t_1$  and  $t_2$  as arguments and applies on them  $|F_1||F_2|$  simple extended SBPs  $(p_i, m)$ , where  $m$  ranges over all 1:1 mappings in the set  $\{(\{f_1\}, \{f_2\}) | f_1 \in F_1, f_2 \in F_2\}$ . The predicate returns the disjunction of these  $|F_1||F_2|$  values as the final output.

*Example 6.* Consider the mapping between sets {Name} in Dataset 2 and {First Name, Last Name} in Dataset 3, in Figure 1. Let the input GBP be *ContainsCommonToken*. The complex extended SBP corresponding to these inputs would be  $\text{ContainsCommonToken}(\{\text{Name}\}, \{\text{First Name}\}) \vee \text{ContainsCommonToken}(\{\text{Name}\}, \{\text{Last Name}\})$ . This complex extended SBP would yield the same result as the simple extended SBP *ContainsCommonToken*({Name}, {Name}) if a new field called *Name* is derived from the merging of the *First Name* and *Last Name* fields in Dataset 3.

An operator like *conjunction* is theoretically possible but may prove restrictive when learning practical schemes. The *disjunction* operator makes complex extended SBPs more expressive than simple extended SBPs, but requires more computation (Section 4.3). Evaluating alternate combine operators is left for future work.

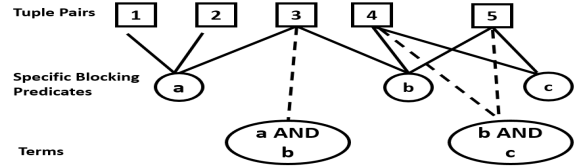


Figure 4: An example showing how  $H_c$  is formed

Finally, (simple or complex) *extended* DNF schemes can be defined in a similar vein as Definition 4, using (simple or complex) extended SBPs as atoms. One key advantage of using disjunction as the combine operator in Definition 6 is that, assuming simple extended SBPs as atoms for *both* simple and complex extended DNF schemes, the scheme remains a positive boolean formula, by virtue of *distributivity* of conjunction and disjunction.

### 4.3 Extending Existing DNF-BSLs

Existing DNF-BSLs [3, 21, 16, 6] rely on similar high-level principles, which is to devise an approximation algorithm for solving the NP-hard optimization problem first formalized by Bilenko et al. [3]. The approximation algorithms are different in that they require different parameters and levels of supervision. These are detailed below and summarized in Table 1. These BSLs were originally designed only for structurally homogeneous tables, with a single field-set  $A$ . We describe their underlying core principles before describing extensions in keeping with the formalism in Section 4.2.

Assume a set of GBPs  $G$ . The *core* of all approximation algorithms would first construct a *search space* of SBPs  $H$  by forming the cross product of  $G$  and  $A$ . The goal of the algorithm is to choose a subset  $H' \subseteq H$  such that the optimization condition<sup>11</sup> laid out by Bilenko et al. is satisfied, at least approximately [3]. The condition assumes that training sets of duplicates  $D$  and non-duplicates  $N$  are provided. Intuitively, the condition states that the *disjunctive* blocking scheme formed by taking the disjunction of SBPs in  $H'$  *covers* (see Definition 4) at least  $\epsilon|D|$  duplicates, while covering the *minimum* possible non-duplicates [3]. Note that  $\epsilon$  is a parameter common to all four systems<sup>12</sup> in Table 1.

In order to learn a *DNF* scheme (as opposed to just disjunctive), a *beam search* parameter  $k$  (also common to all

<sup>11</sup>We describe the condition here intuitively, and formally state it in the Appendix.

<sup>12</sup> $\epsilon$  was designated as *min\_thresh* in the original System 1 paper [21], and  $\sigma$  in the System 3 paper [6].



Table 1: DNF-BSL Systems

ID	System	Parameters	Supervision
1	Michelson and Knoblock [21]	$\epsilon, k$	Supervised
2	Bilenko et al. [3]	$\epsilon, \eta, k$	Supervised
3	Cao et al. [6]	$s, \epsilon, \tau, \alpha, k$	Semi-sup.
4	Kejriwal and Miranker [16]	<b>Generator:</b> $c, ut, lt, d, nd$ <b>Learner:</b> $\epsilon, \eta, k$	Unsup.
5	Algorithm 1 <i>herein</i>	$\kappa, k$	Unsup.

four systems) is required. This parameter is used to *supplement* the original set  $H$  with *terms*, to obtain a new set  $H_c$ . This combinatorial process is demonstrated in Figure 4.

$H$  originally consists of the SBPs  $a, b$  and  $c$ . These SBPs cover some tuple pairs (TPs). Suppose  $k = 2$ . A term of size 2 is formed by checking if any TP is covered by (at least) two SBPs. For example, TP-3 is covered by SBPs  $a$  and  $b$ , and hence, also covered by the term  $a \wedge b$ . For  $k > 2$ , terms from size 2 to size  $k$  are *recursively* added to  $H$ ; the final supplemented set is denoted as  $H_c$ . Note that for  $|H|$  predicates, building  $H_c$  takes  $O(|H|^k)$  time *per* TP. Given the exponential dependence on  $k$  and diminishing returns, previous results capped  $k$  at 2 [16, 3]. If  $k = 1$ ,  $H_c = H$ .

The set  $H' \subseteq H_c$  that is now chosen by the approximation scheme would potentially consist of terms and SBPs, with their disjunction yielding a  $k$ -DNF scheme<sup>13</sup>.

While System 1 only requires  $\epsilon$  and  $k$  as its parameters, Systems 2 and 4 *prune* their search spaces by removing all SBPs and terms from  $H_c$  that cover more than  $\eta|N|$  non-duplicates. Note that this step *heuristically* improves both<sup>14</sup> quality and run-time. It comes at the risk of failure, since if the search space is pruned excessively (by high  $\eta$ ), it may become impossible to cover at least  $\epsilon|D|$  duplicates. Systems 3 and 4 require less supervision but significantly more *parameter tuning*, given they rely on many more parameters.

Systems 1-3 rely on different<sup>15</sup> *Set Covering* (SC) variants [10]. All three systems can be extended by constructing a search space of complex extended SBPs using  $G$  and the mappings set  $Q$ , instead of  $G$  and a field set  $A$ . The underlying SC approximations operate in this abstract search space to choose the final set  $H'$ . An extended DNF scheme is formed by a disjunction of (extended) elements in  $H'$ .

Modifying System 4 is problematic because the system is *unsupervised* and runs in *two* phases. The first phase (denoted as *generator*) generates a noisy training set, and the second phase (denoted *learner*) performs feature-selection on the noisy set to output a DNF blocking scheme. The feature-selection based learner is similar to Systems 1-3 and *can* be extended. Unfortunately, the generator *explicitly* assumes homogeneity, and cannot be directly extended to generate training examples for heterogeneous datasets. This implies that the DNF-BSL component of the proposed generic pipeline in Figure 2(a) *cannot* be instantiated with an existing unsupervised system.

In the event that a schema matcher (and thus,  $Q$ ) is unavailable in Figure 2(a), we present a *fall-back option*.

Specifically, we build a set  $Q$  of *all* 1:1 mappings,  $|Q| = |A_1||A_2|$ . Recall  $A_i$  is the field set of dataset  $i$ . We denote the constructed set  $H$  of SBPs as *simple exhaustive*. Note that for the set of *all* mappings ( $\approx 2^{|A_1|+|A_2|}$ ), the constructed set  $H$  (denoted *complex exhaustive*) is not computationally feasible for non-trivial cases. Even the simple exhaustive case is only a fall-back option, since a *true* set of 1:1 mappings  $Q$  would be much smaller<sup>16</sup> than this set.

## 5. AN UNSUPERVISED INSTANTIATION

A key question to ask is whether the generic pipeline can be instantiated in an *unsupervised* fashion. As we showed earlier, existing DNF-BSLs that can be extended require some form of supervision. An unsupervised heterogeneous DNF-BSL is important because, *in principle*, it enables a fully unsupervised *ER workflow* in both the relational and Semantic Web communities. As the surveys by Elmagarmid et al. and Ferraram et al. note, unsupervised techniques for the *second* ER step do exist already [12, 13]. A second motivation is the observation that existing unsupervised and semi-supervised homogeneous DNF-BSLs (Systems 3-4) require considerable parameter tuning. Parameter tuning is increasingly being cited as an important algorithmic issue, in applications ranging from schema matching [18] to generic machine learning [11]. *Variety* in Big Data implies that algorithm design cannot discount parameter tuning.

We propose an unsupervised instantiation with a *new* DNF-BSL that requires only *two* parameters. In Table 1, only the supervised System 1 requires two parameters. The schematic of the unsupervised instantiation (of the generic pipeline in Figure 2(a)) is shown in Figure 2(b). We use the *existing* schema matcher, *Dumas*, in the instantiated pipeline [4]. *Dumas* outputs 1:1 field mappings by first using a *duplicates generator* to locate tuple pairs with high *cosine similarity*. In the second step, *Dumas* uses *Soft-TFIDF* to build a *similarity matrix* from each generated duplicate. If  $n$  duplicates are input to the second step,  $n$  similarity matrices are built and then averaged into a single similarity matrix. The *assignment problem* is then solved by invoking the *Hungarian Algorithm* on this matrix [19]. This results in exactly  $\min(|A_1|, |A_2|)$  1:1 field mappings (the set  $Q$ ) being output.

In addition to using  $Q$ , we recycle the noisy duplicates of *Dumas* and pipe them into Algorithm 1. Note that *Dumas* does not generate non-duplicates. We address this issue in a novel way, by *permuting* the generated duplicates set  $D$ . Suppose that  $D$  contains  $n$  tuple pairs  $\{(r_1, s_1), \dots, (r_n, s_n)\}$ , with each  $r, s$  respectively from datasets  $R_1, R_2$ . By randomly permuting the pairs in  $D$ , we *heuristically* obtain non-duplicate pairs of the form  $(r_i, s_j)$ ,  $i \neq j$ . Note that (at most)  $n!$  distinct permutations are possible. For balanced supervision, we set  $|N| = |D|$ , with  $N$  the permutation-generated set.

Empirically, the permutation is expected to yield a precise  $N$  because of observed *duplicates sparsity* in ER datasets [9, 16]. This sparsity is also a key tenet underlying the blocking procedure itself. If the datasets were dense in duplicates, blocking would not yield any savings.

Algorithm 1 shows the pseudocode of the extended DNF BSL. Inputs to the algorithm are the piped *Dumas* outputs,  $D$  and  $Q$ . To learn a blocking scheme from these inputs, two parameters  $k$  and  $\kappa$  need to be specified. Similar to

<sup>13</sup>A  $k$ -DNF formula has at most  $k$  literals in each term.

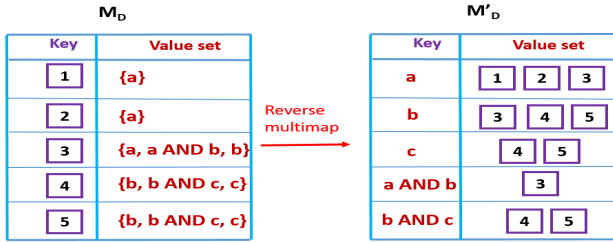
<sup>14</sup>Since  $H_c$  now contains only a few, high-quality elements.

<sup>15</sup>Surveyed briefly in the Appendix.

<sup>16</sup>At most  $\min(|A_1|, |A_2|)$

**Algorithm 1** Learn Extended k-DNF Blocking Scheme

**Input :** Set  $D$  of duplicate tuple pairs, Set  $Q$  of mappings  
**Parameters :** Beam search parameter  $k$ , SC-threshold  $\kappa$   
**Output :** Extended DNF Blocking Scheme  $\mathcal{B}$   
**Method :** //Step 0: Construct sets  $N$  and  $H$   
 Permute pairs in  $D$  to obtain  $N$ ,  $|N| = |D|$   
 Construct set  $H$  of simple extended SBPs using set  $G$  of GBPs and  $Q$   
 Supplement set  $H$  to get set  $H_c$  using  $k$   
 //Step 1: Build Multimaps  $M'_D$  and  $M'_N$   
 Construct  $M_D = \langle X, H_X \rangle$ ,  $X$  is a tuple pair in  $D$ ,  $H_X \subseteq H_c$  contains the elements in  $H_c$  covering  $X$   
 Repeat previous step to build  $M_N$  for tuple pairs in  $N$   
 Reverse  $M_D$  and  $M_N$  to respectively get  $M'_D$  and  $M'_N$   
 //Step 2: Run approximation algorithm  
**for all**  $X \in \text{keyset}(M'_D)$  **do**  
   Score  $X$  by using formula  $|M'_D(X)|/|D| - |M'_N(X)|/|N|$   
   Remove  $X$  if  $\text{score}(X) < \kappa$   
**end for**  
 Perform W-SC on keys in  $M'_D$  using Chvatal's heuristic, weights are *negative* scores  
 //Step 3: Construct and output DNF blocking scheme  
 $\mathcal{B} :=$  Disjunction of chosen keys  
 Output  $\mathcal{B}$



**Figure 5:** Step 1 of Algorithm 1, assuming the information in Figure 4

(extended) Systems 1-3 in Table 1,  $G$ ,  $Q$  and  $k$  are used to construct the search space,  $H_c$ . Note that  $G$  is considered the algorithm's *feature space*, and is not a dataset-dependent input (or parameter). Designing an *expressive*  $G$  has computational and qualitative effects, as we empirically demonstrate. We describe the GBPs included in  $G$  in Section 6.

Step 0 in Algorithm 1 is the permutation step just described, to generate the non-duplicates set  $N$ .  $G$  and  $Q$  are then used to construct the set  $H$  of simple extended<sup>17</sup> SBPs (Definition 5), with  $|H| = |G||Q|$ .  $H$  is supplemented (using parameter  $k$ ) to yield  $H_c$ , as earlier described in Section 4.3.

Step 1 constructs *multimaps*<sup>18</sup> on which Set Covering (SC) is eventually run. As a first logical step, multimaps  $M_D$  and  $M_N$  are constructed. Each tuple pair (TP) in  $D$  is a *key* in  $M_D$ , with the SBPs and terms in  $H_c$  covering that TP comprising the *value set*.  $M_D$  is then *reversed* to yield  $M'_D$ .  $M'_N$  is built analogously. Figure 5 demonstrates the procedure, assuming  $D$  contains TPs 1-5, covered as shown in Figure 4. The time complexity of building (both)  $M'_D$  and  $M'_N$  is  $O(|H|^k(|D| + |N|))$ .

In Step 2, each key is first scored by calculating the dif-

ference between the *fractions* of covered duplicates and non-duplicates. A threshold parameter,  $\kappa$ , is used to remove the SBPs and terms that have low scores. Intuitively,  $\kappa$  tries to *balance* the conflicting needs of previously described parameters,  $\epsilon$  and  $\eta$ , and reduce tuning effort. The range of  $\kappa$  is  $[-1, 1]$ . An advantage of the parameter is that it has an intuitive interpretation. A value close to 1.0 would indicate that the user is confident about low noise-levels in inputs  $D$  and  $Q$ , since high  $\kappa$  implies the existence of elements in  $H_c$  that cover many positives and few negatives. Since many keys in  $M'_D$  are removed by high  $\kappa$ , this also leads to computational savings. However, setting  $\kappa$  too high (perhaps because of misguided user confidence) could potentially lead to excessive purging of  $M'_D$ , and subsequent algorithm failure. Experimentally, we show that  $\kappa$  is easily tunable and even high values of  $\kappa$  are robust to noisy inputs.

*Weighted Set Covering* (W-SC) is then performed using Chvatal's algorithm<sup>19</sup> [10], with each key in  $M'_D$  acting as a *set* and the tuple pairs covered by all keys as elements of the *universe set*  $\mathcal{U}$ . For example, assuming all SBPs and terms in the keyset of  $M'_D$  in Figure 5 have scores above  $\kappa$ ,  $\mathcal{U} = \{1, 2, 3, 4, 5\}$ . Note that *only*  $M'_D$  is pruned (using  $\kappa$ ) and also, W-SC is performed only on  $M'_D$ .  $M'_N$  only aids in the score calculation (and subsequent pruning process) and may be safely purged from memory before W-SC commences.

W-SC needs to find a subset of the  $M'_D$  *keyset* that covers *all* of  $\mathcal{U}$  and with *minimum* total weight. For this reason, the weight of each set is the *negative* of its calculated score. Given that sets chosen by W-SC actually represent SBPs or terms, their disjunction is the k-DNF blocking scheme.

Under plausible<sup>20</sup> complexity assumptions, Chvatal's algorithm is essentially the *best-known* polynomial-time approximation for W-SC [23]. For example, Bilenko et al. used Peleg's approximation to *Red-Blue SC* [22, 7], which is known to have worse bounds [7]. The proposed DNF-BSL has the strongest theoretical approximation guarantee of all systems in Table 1.

## 6. EXPERIMENTS

### 6.1 Metrics

The quality and efficiency of blocking is evaluated by the special metrics, *Reduction Ratio* (RR), *Pairs Completeness* (PC) and *Pairs Quality* (PQ) [9]. Traditional metrics like precision and recall do not apply to blocking since it is a *pre-processing* step, and its output is not the final ER output. To define RR, which intuitively measures *efficiency*, consider the full set of pairs  $\Omega$  that would be generated in the *absence* of blocking. Specifically,  $\Omega$  is the set  $\{(r, s) | r \in R_1, s \in R_2\}$ . RR is then given by the formula,  $1 - |\Gamma|/|\Omega|$ . Given the sparsity of duplicates in real-world datasets [9], RR should ideally be close to, but not precisely, 1.0 (unless  $\Gamma = \phi$ ).

Denote the set of all duplicate pairs, or the *ground-truth*, as  $\Omega_m$ . Similarly, consider the set of duplicates *included* in the candidate set  $\Gamma_m = \Gamma \cap \Omega_m$ . The metric PC quantifies the *effectiveness* of the blocking scheme by measuring *coverage* of  $\Gamma_m$  with respect to  $\Omega_m$ . Specifically, it is given by the formula,  $|\Gamma_m|/|\Omega_m|$ . Low PC implies that recall on *overall* ER will be low, since many duplicates did not share a block to begin with.

<sup>17</sup>Since Dumas only outputs 1:1 mappings

<sup>18</sup>Multimap keys reference multiple values (or a *value set*).

<sup>19</sup>We include Chvatal's algorithm in the SC Appendix survey.

<sup>20</sup> $P \subset NP$

PC and RR together express an efficiency-effectiveness tradeoff. The metric PQ is sometimes used to measure how *dense* the blocks are in duplicates, and is given by  $|\Gamma_m|/|\Gamma|$ . PQ has not been reported in recent BSL literature [3],[16]. One reason is that PQ can be equivalently expressed as  $c \cdot PC / (1 - RR)$ , where  $c$  is  $|\Omega_m|/|\Omega|$ . When *comparing* two BSLs, PQ can therefore be expressed wholly in terms of PC and RR. We do not consider PQ further in this paper.

## 6.2 Datasets

The heterogeneous test suite of six dataset *pairs* (and nine individual datasets) is summarized in Table 2. The suite spans over four domains and the three kinds of heterogeneity discussed in the paper. All datasets are from real-world sources. We did not curate these files in any way, except for serializing RDF datasets as property tables (instead of triples-sets). The serializing was found to be near-instantaneous ( $< 1$  second) in all cases, with negligible run-time compared to the rest of the pipeline.

*Dataset Pairs* (DPs) 1 and 2 are the RDF benchmarks in the 2010 *instance-matching* track<sup>21</sup> of OAEI<sup>22</sup>, an annual Semantic Web initiative. Note that an earlier *tabular* version of DP 1 is also popular in *homogeneous* ER literature [9].

DPs 3,5 and 6 describe *video game* information. DP 6 has already been used as a test case in a previous schema matching work [26]. *vgchartz* is a tabular dataset taken from a reputable charting website<sup>23</sup>. *DBpedia* contains 48,132 *triples* extracted from DBpedia<sup>24</sup>, and has four (three<sup>25</sup> properties and *subject*) fields and 16,755 tuples in property table form. Finally, *IBM* contains user-contributed data extracted from the *Many Eyes* page<sup>26</sup>, maintained by IBM Research.

DP 4 describes US *libraries*. *Libraries 1* was from a *Point of Interest* website<sup>27</sup>, and *Libraries 2* was taken from a US government listing of libraries.

DPs 2 and 4 contain  $n:m$  ground-truth schema mappings, while the others only contain 1:1 ground-truth mappings.

## 6.3 Baselines

Table 1 listed existing DNF-BSLs, with Section 4.3 describing how the extended versions fit into the pipeline. We extend two state-of-the-art systems as baselines:

**Supervised Baseline:** System 2 was chosen as the supervised baseline [3], and favored over System 1 [21] because of better reported empirical performance on a common benchmark employed by both efforts. Tuning the parameter  $\eta$  leads to better blocking schemes, making System 2 a state-of-the-art supervised baseline.

**Semi-supervised Baseline:** We *adapt* System 4 as a *semi-supervised* baseline by feeding it the same noisy duplicates generated by Dumas as fed to the proposed learner, as well as a *manually* labeled negative training set. The learner of System 4 uses *feature selection* and was shown to be empirically competitive with *supervised* baselines [16]. In contrast, System 3 did not evaluate its results against System 2. Also, the learner of System 4 requires three param-

eters, versus the five of System 3. Finally, the three System 4 parameters are *comparable* to the corresponding System 2 parameters, and evaluations in the original work showed that the learner is robust to minor parameter variations [16]. For these reasons, the feature-selection learner of System 4 was extended to serve as a semi-supervised baseline.

## 6.4 Methodology

In this section, we describe experimental methodology. Experimental *results* will be presented in Section 7.

### 6.4.1 Dumas

Given that the test suite is larger and more heterogeneous in this paper than in the original Dumas work [4], we perform two *preliminary* experiments to evaluate Dumas. Dumas was briefly described in Section 5.

**Preliminary Experiment 1:** We evaluate the performance of Dumas’s *duplicates generator*. The generator uses TF-IDF to retrieve the highest scoring duplicates *in order*. Suppose we retrieve  $t$  duplicates (as an ordered list). Denote, for any  $k \leq t$ , the number of true positives in sub-list  $[1 \dots k]$  as  $d(k)$ . Define *Precision@k* as  $d(k)/k$  and *Recall@k* as  $d(k)/|\Omega_m|$ , where  $\Omega_m$  is the ground-truth set of all duplicates. We plot *Precision@k* against *Recall@k* for all DPs to demonstrate the precision-recall tradeoff. To obtain a full set of data points, we set  $t$  at 50,000 for all experiments, and calculate *Precision@k* and *Recall@k* for  $k \in \{1 \dots t\}$ .

**Preliminary Experiment 2:** Although  $t$  was set to a high value in the previous experiment, Dumas requires only a few top pairs (typically 10-50) for the schema matching step [4]. To compute the similarity matrix from  $t$  pairs, Dumas uses *Soft* TF-IDF, which requires an *optional* threshold  $\theta$ . For a given DP, denote  $Q'$  as the ground-truth set of schema mappings,  $Q$  as the set of Dumas mappings, and  $Q_m \subseteq Q$  as the set of *correct* Dumas mappings. Define *precision* (on this task) as  $|Q_m|/|Q|$  and *recall* as  $|Q_m|/|Q'|$ . In this experiment, we set  $t$  and  $\theta$  to *default* values of 50 and 0.5 respectively and report the results. We also vary these parameters and describe performance differences.

### 6.4.2 DNF BSL Parameter Tuning

We describe parameter tuning methodology. Note that the beam search parameter  $k$  (see Table 1) is not technically tuned, but *set*, incumbent on the experiment (Section 6.4.3).

**Baseline Parameters:** Both baseline parameters are tuned in similar ways. We do an exhaustive parameter *sweep* of  $\epsilon$  and  $\eta$ , with values in the range of 0.90 – 0.95 and 0.0 – 0.05 (respectively) typically maximizing baseline performance. Low  $\eta$  maximizes RR, while high  $\epsilon$  maximizes PC. Although extreme values, with  $\eta = 0$  and  $\epsilon = 1$ , are *expected* to maximize performance, they led to failure<sup>28</sup> on all test cases. This demonstrates the necessity of proper parameter tuning. Fewer parameters imply less tuning, and faster system deployment.

**SC-threshold  $\kappa$ :** The single parameter  $\kappa$  of the proposed method was tuned on the smallest test case (DP 1) and found to work best at 0.9. To test the *robustness* of  $\kappa$  to different domains, we fixed it at 0.9 for all experiments.

**$|D|$  and  $|N|$ :** We emulate the methodology of Bilenko et al. in setting the numbers of positive and negative samples input to the system. Bilenko et al. input 50% of true positives and an *equal* number of negatives to train their

<sup>28</sup> $\epsilon|D|$  duplicates could not be covered (see Section 4.3).

<sup>21</sup><http://oei.ontologymatching.org/2013/#instance>

<sup>22</sup>Ontology Alignment Evaluation Initiative

<sup>23</sup>[vgchartz.com](http://vgchartz.com)

<sup>24</sup>[dbpedia.org](http://dbpedia.org)

<sup>25</sup>*genre, platform and manufacturer.*

<sup>26</sup>[www-958.ibm.com/software/data/cognos/manyeyes/datasets](http://www-958.ibm.com/software/data/cognos/manyeyes/datasets)

<sup>27</sup><http://www.poi-factory.com/poifiles>



Table 2: Details of dataset pairs. The notation, where applicable, is (first dataset) /  $\times$  (second dataset)

ID	Dataset Pairs	Fields	Total Entity Pairs	Duplicate Pairs	Data Model
1	Restaurant 1 / Restaurant 2	8/8	$339 \times 2256 = 764,784$	100	RDF/RDF
2	Persons 1 / Persons 2	15/14	$2000 \times 1000 = 2$ million	500	RDF/RDF
3	IBM/vgchartz	12/11	$1904 \times 20,000 \approx 38$ million	3933	Tabular/Tabular
4	Libraries 1 / Libraries 2	5/10	$17,636 \times 26,583 \approx 469$ million	16,789	Tabular/Tabular
5	IBM/DBpedia	12/4	$1904 \times 16,755 \approx 32$ million	748	Tabular/RDF
6	vgchartz/DBpedia	11/4	$20,000 \times 16,755 \approx 335$ million	10,000	Tabular/RDF

system [3]. Let this number be  $n (= |D|, |N|)$  for a given test suite. For example,  $n = 50$  for DP 1. For fairness, we use these numbers for the semi-supervised baseline and proposed system also. We retrieve the top  $n$  pairs from the Dumas generator and input the pairs to both systems as  $D$ . Additionally, we provide  $n$  labeled non-duplicates (as  $N$ ) to the semi-supervised baseline. In subsequent experiments, the dependence of the proposed system on  $n$  is tested.

### 6.4.3 Extended DNF BSL

We describe the evaluation of the extended DNF BSLs.

**Set  $G$  of GBPs:** Bilenko et al. first proposed a set  $G$  that has since been adopted in future works [3, 6, 16]. This original set contained generic token-based functions, numerical functions and character-based functions [3]. No recent work attempted to supplement  $G$  with more expressive GBPs. In particular, *phonetic* GBPs such as *Soundex*, *Metaphone* and *NYSIIS* were not added to  $G$ , despite proven performance benefits [8]. We make an empirical contribution by supplementing  $G$  with all nine phonetic features implemented in an open-source package<sup>29</sup>. For fairness, the same  $G$  is always input to all learners in the experiments below. Results on Experiment 2 will indirectly demonstrate the benefits of supplementing  $G$ . A more detailed description of the original (and supplemented)  $G$  is provided in the Appendix.

**Experiment 1:** To evaluate the proposed *learner* against the baselines, we input the same  $Q$  (found in Preliminary Experiment 2) to *all* three systems. The systems learn the DNF scheme by choosing a subset  $H' \subseteq H_c$  of SBPs and supplemented terms, with  $H_c$  constructed from  $G$ ,  $Q$  and  $k$  (Section 4.3). We learn only *disjunctive* schemes by setting  $k$  to 1 in this experiment.

**Experiment 2:** We repeat Experiment 1 but with  $k = 2$ . We mentioned in Section 4.3 that complexity is *exponential* in  $k$  for all systems in Table 1. Because of this exponential dependence, it was not possible to run the experiment for  $k = 2$  on all DPs. We note which DPs presented problems, and why. Note that, if  $G$ ,  $Q$  and the training sets are fixed, increasing  $k$  seems to be the only feasible way of improving blocking quality. However,  $G$  is more expressive in this paper. Intuitively, we expect the difference across Experiments 1 and 2 to be narrower than in previous work.

**Experiment 3:** In a third set of experiments, we evaluate how blocking performance varies with  $Q$ . To the baseline methods, we input the set of (possibly  $n:m$ ) ground-truth mappings  $Q'$  while the Dumas output  $Q$  is retained for the proposed learner. The goal is to evaluate if extended DNF-BSLs are *sensitive*, or if noisy 1:1 matchers like Dumas suffice for the end goal.

**Experiment 4:** We report on run-times and show per-

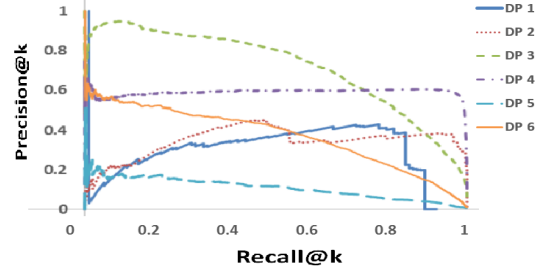


Figure 6: Dumas duplicates-generation results, for  $k \in \{1, 2, \dots, 50000\}$

formance variations of the proposed system with the number of provided duplicates,  $n$ . In industrial settings with an unknown ground-truth,  $n$  would have to be estimated. An important question is if we can rely on getting good results with *constant*  $n$ , despite DP heterogeneity.

**Statistical Significance:** We conduct experiments in ten runs, and (where relevant) report statistical significance levels using the paired sample Student’s t-distribution. On blocking metrics, we report if results are not significant (NS), weakly significant (WS), significant (SS) or highly significant (HS), based on whether the p-value falls within brackets [1.0, 0.1), (0.05, 0.1], (0.01, 0.05] and [0.0, 0.01] respectively. As for the choice of samples, we always *individually* paired PC and RR of the proposed system against the baseline that achieved a *better* average on the metric.

**Implementation:** All programs were implemented in Java on a 32-bit Ubuntu virtual machine with 3385 MB of RAM and a 2.40 GHz Intel 4700MQ i7 processor.

## 7. RESULTS AND DISCUSSION

### 7.1 Dumas

**Preliminary Experiment 1:** Figure 6 shows the results of Dumas’s duplicates-generator. Except for DP 3, precision on all cases seems inadequate even at low recall levels. Although recall of 100% is eventually attained on most DPs, the price is (near) 0% precision. Closer inspection of the results showed that many false positives got ranked at the top. We discuss the implications of these noisy results shortly.

**Preliminary Experiment 2:** Table 3 shows the schema mapping results retrieved from Dumas with parameters set at  $t = 50$  and  $\theta = 0.5$ . These default values were found to maximize performance in *all* cases, in agreement with similar values in the original work [4]. We also varied  $t$  from 10 to 10,000 and  $\theta$  from 0 to 0.9. Performance slightly declined (by at most 5%) for some DPs when  $t < 50$ , but remained

<sup>29</sup>org.apache.commons.codec.language

**Table 3: Best Results of Dumas Schema-Matcher**

Dataset Pair	Precision	Recall
1	100%	87.5%
2	92.86%	86.67%
3	91%	100%
4	75%	33.33%
5	100%	100%
6	100%	100%

otherwise constant across parameter sweeps. This confirms that Dumas is quite robust to  $t$  and  $\theta$ . One *disadvantage* of Dumas is that it is a 1:1 matcher. This explains the lower recall numbers on DPs 2 and 4, which contain n:m mappings. In Experiment 3, we test if this is problematic by providing the ground-truth set  $Q'$  to baselines, and comparing results.

**Discussion:** An important point to note from the (mostly) good results in Table 3 is that generator accuracy is not always *predictive* of schema mapping accuracy. This robustness to noise is always an important criteria in pipelines. If noise *accumulates* at each step, the final results will be qualitatively weak, possibly meaningless. Since Dumas’s matching component is able to compensate for generator noise, it is a good empirical candidate for unsupervised 1:1 matching on typical ER cases. Preliminary Experiment 1 also shows that on real-world ER problems, a simple measure like TF-IDF is not appropriate (by itself) for solving ER. The generator noise provides an interesting test for the extended DNF-BSLs. Since both the mappings-set  $Q$  and top  $n$  generated duplicates (the set  $D$ ) output by Dumas are piped to the learner, there are *two* potential sources of noise.

Finally, given that the proposed system (in subsequent experiments) *permutes* the Dumas-output  $D$  (Algorithm 1) to generate  $N$ , we tested the accuracy of the permutation procedure in a follow-up experiment. With  $|D|$  ranging from 50 to 10,000 in increments of 50, we generated  $N$  of equal size and calculated *non-duplicates accuracy*<sup>30</sup> of  $N$  over ten random trials per value. In all cases, accuracy was 100%, showing that the permutation heuristic is actually quite strong.

## 7.2 Extended DNF BSL

**Experiment 1:** Table 4 shows BSL results on all six DPs. The high overall performance explains the recent popularity of DNF-BSLs. Using the extended DNF hypothesis space for blocking schemes allows the learner to compensate for the two sources of noise earlier discussed. Overall, considering statistically significant results, the supervised method typically achieves better RR, but PC is (mostly) equally high for all methods, with the proposed method performing the best on DP 4 (the largest DP) and the supervised baseline on DP 2, with high significance. We believe the former result was obtained because the proposed method has the strongest approximation bounds out of all three systems, and that this effect would be most apparent on large DPs. Importantly, low standard deviation (often 0) is frequently observed for all methods. The DNF-BSLs prove to be quite *deterministic*, which can be important when replicating results in both research and industrial settings.

**Experiment 2:** Next, we evaluated if  $k = 2$  enhances BSL performance and justifies the exponentially increased

cost. With  $k = 2$ , only DPs 1 and 5 were found computationally feasible. On the other DPs, the program either ran out of RAM (DPs 4,6), or did not terminate after a long<sup>31</sup> time (DPs 2,3). The former was observed because of high  $n$  and the latter because of the large number of *fields* (see Table 2). Setting  $k$  beyond 2 was computationally infeasible even for DPs 1 and 5. Furthermore, results on DPs 1 and 5 showed *no* statistical difference compared to Experiment 1, even though run-times went up by an approximate factor of 16 (for both DPs).

**Experiment 3:** We provided the ground-truth set  $Q'$  to baseline methods (and with  $k$  again set to 1), while retaining  $Q$  for the proposed method. Again, we did not observe any statistically significant difference in PC or RR for either baseline method. We believe this is because the cases for which  $Q'$  would most likely have proved useful (DPs 2 and 4, which contain n:m mappings that Dumas cannot output) already perform well with the Dumas-output  $Q$ .

**Experiment 4:** Theoretically, run-time of Algorithm 1 was shown to be  $O(|H|^k(|D| + |N|))$ ; the run-times of other systems in Table 1 are similar. Empirically, this has never been demonstrated. For  $k = 1$ , we plot the run-time data collected from Experiment 1 runs on DPs 1-6 (Figure 7(a)). The trend is fairly linear, with the supervised system slower for smaller inputs, but not larger inputs. The dependence on  $|Q|$  shows why a schema matcher is necessary, since in its absence, the *simple exhaustive* set is input (Section 4.3).

As further validation of the theoretical run-time, Figure 7(b) shows the linear dependence of the proposed system on  $|D|$ . Again, the trend is linear, but the slope<sup>32</sup> depends on the *schema heterogeneities* of the individual DPs. For example, DPs 2 and 3, the largest datasets in terms of *fields* (Table 2), do not scale as well as the others.

Figure 7(c) shows an important *robustness* result. We plot the PC-RR *f-score*<sup>33</sup> of the proposed system against  $|D|$ . The first observation is that, even for small  $|D|$ , performance is already high except on DP 2, which shows a steep increase when  $|D| \approx 100$ . On all cases, maximum performance is achieved at about  $|D| \approx 700$  and the f-score curves flatten subsequently. This is qualitatively similar to the robustness of Dumas to small numbers of positive samples.

Figure 7(c) also shows that the proposed method is robust to *overestimates* of  $|D|$ . DP 1, for example, only has 100 true positives, but it continues to perform well at much higher  $|D|$  (albeit with a slight dip at  $|D| \approx 700$ ).

**Discussion:** We earlier stated, when discussing the preliminary experimental results, that an extended DNF-BSL can only integrate well into the pipeline if it is robust to noise from previous steps. Previous research has noted the overall robustness of DNF-BSLs. This led to the recent emergence of a homogeneous unsupervised system [16], adapted here as a semi-supervised baseline. Experiment 1 results showed that this robustness also carries over to *extended* DNF-BSLs. High overall performance shows that the pipeline can accommodate heterogeneity, a key goal of this paper.

Experiment 2 results demonstrate the advantage of having an expressive  $G$ , which is evidently more viable than increasing  $k$ . On DPs 1 and 5 (that the systems succeeded

<sup>31</sup>Within a factor of 20 of the average time taken by the system for the  $k = 1$  experiment (for that DP).

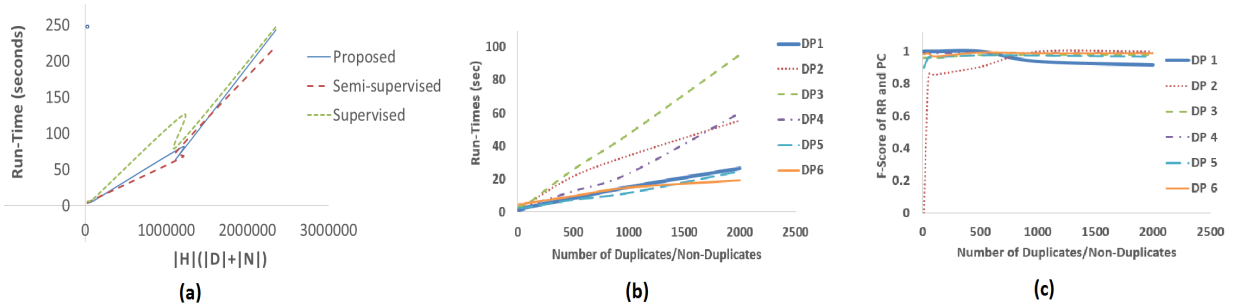
<sup>32</sup>The slope is the hidden constant in the asymptotic notation.

<sup>33</sup> $\frac{2 \cdot RR \cdot PC}{RR + PC}$

<sup>30</sup> $1 - |N \cap \Omega_m|/|N|$

**Table 4: Comparative Results of Extended DNF BSLs. Bold values are (at least) weakly significant, with significance levels (WS,SS or HS) in paranthesis**

Dataset Pair (DP)		Proposed Method		Semi-Supervised Baseline		Supervised Baseline	
		PC	RR	PC	RR	PC	RR
DP 1	Average	100%	99.68%	100%	99.68%	100%	98.59%
	Standard Deviation	0%	0%	0%	0%	0%	3.41%
DP 2	Average	95%	86.11%	95%	99.23%	<b>99.6% (HS)</b>	<b>99.96% (HS)</b>
	Standard Deviation	0%	0%	0%	0%	0%	0%
DP 3	Average	100%	95.47%	100%	95.44%	99.29%	<b>99.99% (HS)</b>
	Standard Deviation	0%	0%	0%	0.01%	0%	0%
DP 4	Average	<b>98.95% (HS)</b>	99.68%	98.43%	<b>99.98% (HS)</b>	98.19%	<b>99.98% (HS)</b>
	Standard Deviation	0.1%	0.007%	0%	0%	0.27%	0.01%
DP 5	Average	100%	92.28%	100%	94.58%	99.46%	<b>97.75% (HS)</b>
	Standard Deviation	0%	0%	0%	1.01%	0.8%	2.36%
DP 6	Average	99.91%	99.69%	99.97%	99.71%	91.09%	<b>99.93% (HS)</b>
	Standard Deviation	0.08%	0.019%	0.07%	0.02%	0.03%	0.004%



**Figure 7: Experiment 4 results. (a) plots run-time trends of all three systems against the theoretical formula, while (b) and (c) respectively plot run-times and f-scores of the proposed system against sample sizes.**

on), no statistically significant differences were observed, despite run-time increasing by a factor of 16. We note that the largest (homogeneous) test cases on which  $k = 2$  schemes were previously evaluated were only about the order of DP 1 (in size). Even with less expressive  $G$ , only a few percentage point performance differences were observed (in PC and RR), with statistical significance *not* reported [16, 3].

To confirm the role of  $G$ , we performed a follow-up experiment where we used the originally proposed  $G$  [3] on DPs 1 and 5, with both  $k = 1$  and  $k = 2$ . We observed lower performance with  $k = 1$  compared to Table 4 results, while  $k = 2$  results were only at par with those results. Run-times with less expressive  $G$  were obviously lower (for corresponding  $k$ ); however,  $k = 2$  run-times were higher (with *less* expressive  $G$ ) than  $k = 1$  run-times with *more* expressive  $G$ . All differences just described were statistically significant (at the 95% level). This validates previous research findings, while also confirming our stated hypothesis about  $G$ .

Experiment 3 results showed that a sophisticated schema matcher is not always necessary for the purpose of learning a blocking scheme. However, the importance of good schema matching goes beyond blocking and even ER. Schema matching is an important step in overall data integration [1]. On noisier datasets, a good n:m schema matcher could make all the difference in pipeline performance, but we leave for future work to evaluate such a case.

The similar run-time trends shown by the various systems in Figure 7(a) also explain why, in Experiment 2, all systems

simultaneously succeeded or failed on a given DP. Even if we replace our DNF-BSL with an extended version from the literature, the exponential dependence on  $k$  remains. Figures 7(a) and (b) also validate theoretical run-time calculations empirically. Previous research on DNF-BSLs did not theoretically analyze (or empirically report) algorithmic run-times and scalability explicitly [16, 3, 6, 21].

Figure 7(c) demonstrates the encouraging qualitative result that only a few (noisy) samples are typically enough for adequate performance. Given enterprise quality requirements, as well as expense of domain expertise, high performance for low  $n$  and minimum parameter tuning is a practical necessity, for industrial deployment. Recall that we retained  $\kappa$  at 0.9 for *all* experiments (after tuning on DP 1), while for the baselines, we had to conduct parameter sweeps for each separate experiment. Combined with results in both Table 4 and Figure 7(c), this shows that the system can be a potential use-case in industry. Combined with previous unsupervised results for the *second* ER step [13, 12], such a use-case would apply both to relational and Semantic Web data as a *fully unsupervised ER workflow*, which has thus far remained elusive.

## 8. CONCLUSION AND FUTURE WORK

In this paper, we presented a generic pipeline for learning DNF blocking schemes on heterogeneous dataset pairs. We proposed an unsupervised instantiation of the pipeline that relies on an existing instance-based schema matcher

and learns blocking schemes using only two parameters. We also showed a novel way of reconciling RDF-tabular heterogeneity by using the *logical* property table representation for building and populating a dynamic property schema for RDF datasets. Finally, we evaluated all techniques on six test cases exhibiting three separate kinds of heterogeneity.

Future research will address further exploration of the property table representation for tabularly mining RDF data. Additionally, refining  $G$  (the set of GBPs) further is a promising, proven method of scalably improving BSL performance. We will also implement a fully unsupervised ER workflow that the proposed unsupervised DNF-BSL *enables*, and evaluate it in a similar fashion.

## 9. REFERENCES

- [1] Z. Bellahsene, A. Bonifati, E. Rahm, et al. *Schema matching and mapping*, volume 20. Springer, 2011.
- [2] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The International Journal on Very Large Data Bases*, 18(1):255–276, 2009.
- [3] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 87–96. IEEE, 2006.
- [4] A. Bilke and F. Naumann. Schema matching using duplicates. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 69–80. IEEE, 2005.
- [5] C. Bizer, T. Heath, and T. Berners-Lee. Linked data—the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.
- [6] Y. Cao, Z. Chen, J. Zhu, P. Yue, C.-Y. Lin, and Y. Yu. Leveraging unlabeled data to scale blocking for record linkage. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 2211, 2011.
- [7] R. D. Carr, S. Doddi, G. Konjevod, and M. V. Marathe. On the red-blue set cover problem. In *Symposium on Discrete Algorithms: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, volume 9, pages 345–353. Citeseer, 2000.
- [8] P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer, 2012.
- [9] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, 24(9):1537–1555, 2012.
- [10] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.
- [11] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.
- [12] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16, 2007.
- [13] A. Ferraram, A. Nikolov, and F. Scharffe. Data linking for the semantic web. *Semantic Web: Ontology and Knowledge Base Enabled Tools, Services, and Applications*, page 169, 2013.
- [14] A. Gal. Why is schema matching tough and what can we do about it? *ACM Sigmod Record*, 35(4):2–5, 2006.
- [15] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep web. *Communications of the ACM*, 50(5):94–101, 2007.
- [16] M. Kejriwal and D. P. Miranker. An unsupervised algorithm for learning blocking schemes. In *Data Mining, 2013. ICDM'13. Thirteenth International Conference on*. IEEE, 2013.
- [17] H.-s. Kim and D. Lee. Harra: fast iterative hashed record linkage for large-scale data collections. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 525–536. ACM, 2010.
- [18] Y. Lee, M. Sayyadian, A. Doan, and A. S. Rosenthal. etuner: tuning schema matching software using synthetic scenarios. *The International Journal on Very Large Data Bases*, 16(1):97–122, 2007.
- [19] L. Lovász and M. D. Plummer. Matching theory. *New York*, 1986.
- [20] Y. Ma, T. Tran, and V. Bicer. Typifier: Inferring the type semantics of structured data. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 206–217. IEEE, 2013.
- [21] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 440. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.
- [22] D. Peleg. Approximation algorithms for the label-cover max and red-blue set cover problems. In *Algorithm Theory-SWAT 2000*, pages 220–231. Springer, 2000.
- [23] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcg characterization of np. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 475–484. ACM, 1997.
- [24] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau Jr, S. Auer, J. Sequeda, and A. Ezzat. A survey of current approaches for mapping of relational databases to rdf. *W3C RDB2RDF Incubator Group Report*, 2009.
- [25] J. F. Sequeda and D. P. Miranker. Ultrawrap: Sparql execution on relational data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 22:19–39, 2013.
- [26] A. Tian, M. Kejriwal, and D. P. Miranker. Schema matching over relations, attributes, and data values. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, page 28. ACM, 2014.
- [27] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *The Semantic Web-ISWC 2009*, pages 650–665. Springer, 2009.
- [28] K. Wilkinson, C. Sayers, H. A. Kuno, D. Reynolds, et al. Efficient rdf storage and retrieval in jena2. In *SWDB*, volume 3, pages 131–150, 2003.

## APPENDIX

### Property Table Algorithms

We describe a procedure for serializing an RDF triples-set as a property table in time  $\Theta(n)$  where  $n$  is the number of triples. The procedure runs in two passes. In a first pass over the set of triples, the procedure would record the distinct properties and subjects in two respective sets,  $S_1$  and  $S_2$ . The *property schema* would then be built using the RDF dataset name and the *field set*  $S_1 \cup \{subject\}$ . The property table would itself be initialized, with the *subject* column populated using  $S_2$ . An *index* to the subject column would also be built, with the subject as key and the row position as value. In the second pass over the triples, the cells of the table are incrementally updated with each encountered triple. An associative map admits an efficient implementation for the second pass.

For completeness, Algorithm 2 shows the inverse serialization; that is, how a property table is converted to a triples-set. Note that both the procedure above and Algorithm 2 are *information-preserving*; no information is lost when we exchange representations.

---

#### Algorithm 2 RDF Property Table to RDF Triples-set

---

**Input :** Property Table with property schema  $P(subject, a_1, \dots, a_n)$  and  $n$  properties  
**Output :** Triples-set  $P'$   
**Method :**  
 Initialize empty triples-set  $P'$   
**for all** tuples  $t$  in property table instance **do**  
   Let  $q = t(subject)$   
   **for all**  $a = a_1 \dots a_n$  **do**  
     **if**  $t(a)$  is null **then**  
       continue  
     **end if**  
     Tokenize  $t(a)$  using ; to obtain (possibly singleton) tokens-set  $T'$   
     **for all** tokens  $t' \in T'$  **do**  
       Add triple  $(q, a, t')$  to  $P'$   
     **end for**  
   **end for**  
**end for**  
 Output  $P'$

---

Algorithm 2 is fairly simple and runs in worst-case time  $O(mn)$ , where  $m$  is the number of subjects in the table, and  $n$  is the total number of properties. We assume that each subject can be bound above by a constant number of object values (per property), a reasonable assumption in real-world cases. If not true, dataset characteristics need to be known before run-time can be bound. We omit a proof that the procedure always yields the same triples-set  $P'$  from which the table was derived, if it was indeed derived from such a table using the two-pass algorithm earlier described. As we noted in Section 4.1, implemented triplestores often *natively* store RDF datasets as *physical* property tables. In a real-world implementation, it is possible to exploit this *systems-level* advantage and use a provided triplestore API to access the physical table in its native form and use it logically [28]. We cited this earlier as a potential advantage in using the property table, and not devising a *new* data structure for resolving RDF-tabular heterogeneity in ER.

### Optimal DNF Schemes

We replicate the optimization condition first formally stated by Bilenko et al. [3]. Assume a (perfectly labeled) training set of *duplicate pairs*  $D$ , and *non-duplicate pairs*  $N$ . Let  $f$  be a blocking scheme from the space  $\mathcal{F}$  of all possible DNF blocking schemes that may be constructed from subsets of the set  $H_c$  of SBPs and terms.  $\mathcal{F}$  provably has cardinality  $2^{|H_c|}$ , since each DNF scheme is merely a *positive* DNF formula  $f$  constructed by treating  $H_c$  as the *atoms-set* (Definition 4). The *optimal* blocking scheme  $f^* \in \mathcal{F}$  satisfies the following objective:

$$f^* = \underset{f}{\operatorname{argmin}} \sum_{\{r,s\} \in N} f(r,s) \quad (2)$$

s.t.

$$\sum_{\{r,s\} \in D} f^*(r,s) \geq |D|\epsilon \quad (3)$$

$D$ ,  $N$  and  $\epsilon$  were defined earlier in the paper. We also stated the meaning of this condition intuitively, which is that the scheme must cover at least a fraction  $\epsilon$  of the set  $D$  while minimizing coverage of non-duplicates in  $N$ . Bilenko et al. were the first to prove that this optimization problem is, in fact, NP-Hard by reducing from Red-Blue Set Covering (see *Set Covering* Appendix section) [3]. We refer the reader to the original work for that proof. Note that the condition is generic in that  $r$  and  $s$  do not have to be from structurally homogeneous datasets, and the proof of Bilenko et al. does not assume such a restriction either [3]. This was one reason we were able to extend their system and use it as an instantiated supervised baseline for the final pipeline module in Figure 2(a). It also implies the natural result that the problem remains NP-Hard for heterogeneous datasets.

### Set Covering

We provide a *generic* description of the Weighted Set Covering (SC) problem, along with Chvatal’s greedy algorithm, which, despite being relatively simple, continues to be the best-known polynomial-time approximation scheme (called PTAS). The weighted instance of SC assumes (as input) a *universe set*  $\mathcal{U}$  with  $n$  elements and a *family* of  $m$  sets  $S = \{S_1, \dots, S_m\}$ , where each  $S_i$  is a subset of  $\mathcal{U}$ . Each set in the family is associated with a *weight*,  $w(S_i)$  for all  $i$  from 1 to  $m$ . Additionally, the condition  $\bigcup_i S_i = \mathcal{U}$  is assumed to hold. The SC problem is to find a subfamily  $\mathcal{C} \subseteq S$  such that the summed weights of all sets in  $\mathcal{C}$  are *minimized* subject to the (mandatory) condition that  $\bigcup_{c \in \mathcal{C}} c = \mathcal{U}$  (denoted as the *covering* condition for the following discussion). The decision version of this problem is known to be NP-Complete, and the optimization version, NP-Hard [10]. This is also true for the non-weighted SC, which reduces to a special case of W-SC with each set assigned equal (usually unit) weight.

Chvatal’s greedy algorithm can be stated simply as follows. Initialize  $\mathcal{C}$  to be the empty set. Iterate over  $S$  till the covering condition is met. In each iteration, pick a (previously unpicked) set  $S_i$  with *maximum* score  $|S_i|/w(S_i)$ . In other words, we greedily pick the set in the family (breaking ties arbitrarily) that covers the most elements per unit weight. It is straightforward to observe that this algorithm is polynomial time; even a simple approach can run in time  $O(mn)$  if the loop body is properly implemented. A linear-time algorithm along the same lines is possible if more ad-



vanced data structures are used. We do not go into details of how to optimize this algorithm; in most cases, an efficient off-the-shelf implementation can be adapted.

In his seminal work (in which he proposed this algorithm), Chvatal also proved that the final (summed) weight of the approximate answer  $\mathcal{C}$  is greater than the optimal answer  $\mathcal{C}^*$  by a factor of (at most)  $H(d)$ , where, for any  $x \in \mathbb{Z}^+$ , the function  $H(x) = \sum_i 1/i$ , for all integers  $i \in [1, x]$  [10]. Note that  $H(x) \leq \ln(x) + 1$ . Here,  $d$  is simply the cardinality of the largest set in  $S$ . Chvatal’s logarithmic approximation ratio remains the best-possible, even decades later [23].

Many variants of SC have been proposed over the decades; an important one is the Red-Blue Set Cover (RB-SC) [22], which is closely related to the supervised method of Bilenko et al. [3]. RB-SC takes a universe set  $\mathcal{U}$  as input, with  $\mathcal{U} = R \cup B$ , where  $R$  is the set of *red* elements and  $B$ , the set of *blue* elements. Note that  $R \cap B$  is empty. Again, we are given a family  $S$  of subsets, but  $S$  is only constrained to cover all of  $B$ , not necessarily the full universe set  $\mathcal{U}$ . RB-SC needs to locate a subfamily  $\mathcal{C}$  such that all blue elements are covered but with the number of *distinct* red elements covered, minimized. Sets in the family are not associated with weights; weighted generalizations of RB-SC are not relevant for this discussion.

RB-SC seems similar to the ordinary SC, but it is ‘harder’ in an approximation sense. The best known approximation ratio for RB-SC, first proved by Peleg, is  $2\sqrt{|\mathcal{U}|\log|B|}$  [22]. In that paper, he proposed an approximation algorithm that was adopted by Bilenko et al. [3].

In Section 5, we explained how to reduce our specific problem to W-SC, by treating  $H_c$  as the family of subsets  $S$ , given that each SBP or term (in  $H_c$ ) covers *tuple pairs*, and with all *duplicate* tuple pairs together comprising the universe set  $\mathcal{U} = D$ . We used the non-duplicates set  $N$  only to calculate weights. On the other hand, Bilenko et al. performed a more *intuitive* reduction to RB-SC, by treating red elements as analogous to non-duplicates and blue elements to duplicates, in a given training set. This direct reduction comes with weak approximation bounds, however, as the discussion above shows. Empirically, we believe that better approximation results led to *at par* PC results (in Experiments 1-2) for the proposed method on four DPs, compared to the supervised baseline, and to *better* PC results (with high significance) on the *largest* DP (4).

## General Blocking Predicates

The set  $G$  of General Blocking Predicates (GBPs) and the parameter  $k$  are used (along with mappings  $Q$  in the heterogeneous case presented in the paper, or the field set  $A$  presented in previous work [16, 21, 3, 6]) to build the *search space* of terms and SBPs,  $H_c$ . Intuitively,  $G$  constitutes the *feature space* of the algorithm and choosing an appropriate  $G$  is an important empirical consideration. Experiment 2 in Section 6, and the described *follow-up* experiment in the subsequent discussion, demonstrate this point. Specifically, the experiments shows that if  $G$  is expressive enough, setting  $k$  to 1 is adequate. Recall that, in this paper, we adapted the original set  $G$ , first proposed by Bilenko et al. and *supplemented* it with phonetic functions found in an open-source package<sup>34</sup>, to make  $G$  more expressive. For completeness, we first provide a brief description of the original set. Note

that all GBPs below are *case-insensitive*.

- (1) **Exact Match:** Returns *True* if input strings exactly match.
- (2) **ContainsCommonToken:** Returns *True* if input strings share a common token, based on common delimiters (such as comma, whitespace and semicolon).
- (3) **ContainsCommonInteger:** Returns *True* if input strings share at least one common integer token. If no token is an integer in a given input string, *False* is returned by default.
- (4) **ContainsCommonOrOffByOneInteger:** Same as above, except integers may be off by one. Note that if *True* is returned by *ContainsCommonInteger*, *True* is also returned for this GBP. This demonstrates that GBPs may be correlated.
- (5-7) **ContainsTokenWithSameNFirstChars:** Returns *True* if the input strings share at least one token with a common  $N$ -character prefix. Implemented with  $N=3,5,7$  to yield three (correlated) GBPs.
- (8-10) **ContainsTokenWithCommonNGram:** Returns *True* if the input strings share a common length- $N$  contiguous subsequence of *tokens*. Implemented with  $N=2,4,6$ .

In total, these yield 10 GBPs. Although these GBPs have been found to work quite well in previous work, including the original work in which they were first proposed [3], a rationale was never provided for why *specifically* each of them were included. We briefly attempt to do so here, based on our experimental observations.

GBPs 1-2 are appropriate for strings that have high token overlap or for alphanumeric codes (in product databases, for example) that tend to match exactly and have high correlation with duplicate classification. GBPs 3-4 are more appropriate for phone numbers, zip codes, street numbers, social security numbers, dates of birth and other numeric quantities that commonly occur in databases. GBPs 5-7 are empirically robust to *data representation* issues; for example, GBP 5 would return *True* for two address strings that spell ‘Avenue’ as *Avenue* or *Ave*. GBPs 8-10 are restrictive versions of GBP 2, and thus, highly *discriminative*. They rarely return *True*, but when they do, it indicates strongly that the input strings are derived from a duplicate pair.

Note that Bilenko et al. included 10 *additional* GBPs that were based on TF-IDF and were appropriate for *homogeneous* datasets [3]. In pilot experiments (and also the Dumas preliminary experiments; see Figure 6), we obtained the unsurprising result that these TF-IDF features had *negative* correlation with heterogeneous BSL performance. These were therefore not included in the  $G$  used in this paper.

In the Apache open-source package, nine phonetic functions are implemented and all were included in the supplemented  $G$ . These are respectively *Caverphone1*, *Caverphone2*, *ColognePhonetic*, *DoubleMetaphone*, *MatchRatingApproachEncoder*, *Metaphone*, *NYSIIS*, *RefinedSoundex* and *Soundex*. Christen provides a good description (and evaluation) of these phonetic encodings in his comprehensive text [8]. Perhaps the most important advantage of phonetic functions is that they are robust to spelling variations (especially in names) that the other GBPs cannot easily accommodate (e.g. Kathryn vs. Catherine).

Finally, it is important to note that each of these GBPs is associated with an *indexing function* (see Definition 1). Typically, the associated indexing functions simply extract some characters, tokens or integers and return the extracted

<sup>34</sup>org.apache.commons.codec.language

elements in a set (GBPs 1-10); similarly, the associated phonetic indexing functions tokenize the string and return a set containing the appropriate phonetic encoding (e.g. Soundex) of each token. GBPs cannot be *arbitrary* boolean functions. As an example, the boolean function *EditDistance*  $< 0.5$  might seem like a legitimate GBP<sup>35</sup>, but it is not evident how to frame it as a *set-intersection* condition on outputs of (some) indexing function, as the original GBP definition (Definition 2) formally requires.

---

<sup>35</sup>It takes two strings as input and returns *True* if the Edit distance is less than 0.5.